

Kurs języka C++

6. Polimorfizm

Spis treści

- ▶ Metody wirtualne
- ▶ Implementacja polimorfizmu
- ▶ Wczesne i późne wiązanie metod wirtualnych
- ▶ Klasy abstrakcyjne
- ▶ Wirtualne destruktory

Składowe funkcje wirtualne

- ▶ Składowe funkcje wirtualne pozwalają na przedefiniowanie w każdej klasie pochodnej funkcji składowych zadeklarowanych w klasie bazowej.
- ▶ Poprzez funkcje wirtualne w programie zapewnione jest wywołanie metody najlepiej odpowiadającej obiektowi.
- ▶ Składowe funkcje wirtualne należy opatrzyć deklaratorem `virtual` (wewnątrz klasy).
- ▶ Składowe funkcje wirtualne nadpisywane w klasach pochodnych należy opatrzyć deklaratorem `override` (wewnątrz klasy).
- ▶ W definicji metody wirtualnej lub wirtualnej i nadpisywanej poza klasą nie używa się ani deklaratora `virtual` ani deklaratora `override`.

Składowe funkcje wirtualne

- Przykład deklaracji klas z metodami wirtualnymi i zwykłymi:

```
class bazowa
{
public:
    void opis_zwykly ();
    virtual void opis_wirtualny ();
};

class pochodna: public bazowa
{
public:
    void opis_zwykly ();
    void opis_wirtualny () override;
};
```

Składowe funkcje wirtualne

► Przykład definicji metod wirtualnych i zwykłych:

```
void bazowa::opis_zwykly()
{ cout << "bazowa::opis_zwykly()" << endl; }
void bazowa::opis_wirtualny()
{ cout << "bazowa::opis_wirtualny()" << endl; }

void pochodna::opis_zwykly()
{ cout << "pochodna::opis_zwykly()" << endl; }
void pochodna::opis_wirtualny()
{ cout << "pochodna::opis_wirtualny()" << endl; }
```

Składowe funkcje wirtualne

- Przykład użycia metod wirtualnych i zwykłych:

```
bazowa *a = new bazowa();  
a->opis_zwykly();  
a->opis_wirtualny();  
// bazowa::opis_zwykly()  
// bazowa::opis_wirtualny()  
  
bazowa *b = new pochodna();  
b->opis_zwykly();  
b->opis_wirtualny();  
// bazowa::opis_zwykly()  
// pochodna::opis_wirtualny()
```

Składowe funkcje wirtualne

- ▶ Funkcja wirtualna musi być zdefiniowana dla klasy, w której po raz pierwszy została zadeklarowana.
- ▶ Funkcji wirtualnej można używać nawet wtedy, gdy z jej klasy nie wyprowadzi się żadnej klasy pochodnej.
- ▶ Klasa pochodna, która nie potrzebuje specjalnej wersji funkcji wirtualnej, nie musi jej dostarczać.
- ▶ Funkcja w klasie pochodnej z tą samą nazwą i z tą samą listą argumentów co funkcja wirtualna w klasie podstawowej **nadpisuje** (ang. *override*) starą wersję funkcji wirtualnej z klasy bazowej.

Polimorfizm

- ▶ Uzyskanie zachowania się funkcji adekwatnego do typu obiektu nazywa się **polimorfizmem** (ang. *polymorphism*).
- ▶ Klasa z funkcjami wirtualnymi nazywa się klasą polimorficzną.
- ▶ Aby zachowanie obiektu było polimorficzne należy się do niego odnosić za pomocą wskaźnika albo referencji.
- ▶ Dzięki polimorfizmowi programy stają się **rozszerzalne** (ang. *extensibility*) - modyfikacja kodu polega na dodaniu nowej klasy bez potrzeby zmian w kodzie istniejącym.

Implementacja zachowań polimorficznych

- ▶ Obiekty klas polimorficznych mają dodatkowe pole identyfikujące typ obiektu.
- ▶ Decyzję o wyborze funkcji do wykonania podejmuje się w trakcie działania programu (jest to tak zwane *późne wiązanie*, w przeciwieństwie do zwykłych funkcji gdzie obowiązuje *wczesne wiązanie*).
- ▶ Każda klasa polimorficzna posiada swoje miejsce w tablicy metod wirtualnych.
- ▶ Polimorfizm jest więc kosztowny (miejsce i czas) - dlatego nie wszystkie metody są wirtualne.

Rezultat funkcji wirtualnej

- ▶ Przy nadpisywaniu funkcji wirtualnej trzeba zachować odpowiedni typ rezultatu:
 - ▶ albo rezultat musi być identyczny,
 - ▶ albo rezultat musi być kowariantny (referencja lub wskaźnik do obiektu tej samej klasy lub do klasy, dla której jest ona jednoznaczna i dostępną klasą bazową).
- ▶ Przykład:

```
owoc* bazowa::fun () { /*...*/ }  
pomelo* pochodna::fun () { /*...*/ }
```

Inne cechy funkcji wirtualnych

- ▶ Funkcja wirtualna w klasie nie może być statyczna.
- ▶ Funkcja wirtualna w klasie nie jest wbudowywana, gdy korzystamy z polimorfizmu.
- ▶ Dostęp do funkcji wirtualnej może być zmieniony w klasach pochodnych (co zależy od sposobu dziedziczenia) - dostęp ten zależy więc tylko od typu wskaźnika albo referencji.
- ▶ Funkcje wirtualne mogą być przyjaciółmi w innych klasach.

Funkcja wirtualna wcześnie związana

- ▶ Funkcja wirtualna będzie wcześniej związana gdy będzie wywołana na rzecz konkretnego obiektu znanego z nazwy:

```
klasa ob;  
// ...  
ob.funwirt();
```

- ▶ Funkcja wirtualna będzie wcześniej związana gdy użyjemy kwalifikatora zakresu:

```
wsk->klasa::funwirt();  
ref.klasa::funwirt();
```

- ▶ Funkcja wirtualna będzie wcześniej związana, gdy wywołamy ją w konstruktorze.

- ▶ Funkcja wirtualna może być wbudowana, gdy korzystamy z wczesnego wiązania funkcji wirtualnych.

Klasy abstrakcyjne

- ▶ Klasy abstrakcyjne służą do definiowania interfejsów (pojęć abstrakcyjnych).
- ▶ Klasa abstrakcyjna zawiera co najmniej jedną abstrakcyjną metodą wirtualną (funkcja czysto wirtualna).
- ▶ Deklaracja metody czysto wirtualnej wygląda następująco:
virtual typ funkcja (lista-argumentów) = 0;
- ▶ Nie trzeba (ale można) podawać definicji metody czysto wirtualnej.
- ▶ W klasach potomnych, które nie mają być klasami abstrakcyjnymi, należy zdefiniować wszystkie odziedziczone metody abstrakcyjne.
- ▶ Klasa potomna, w której nie będą zdefiniowane wszystkie odziedziczone metody abstrakcyjne, będzie nadal klasą abstrakcyjną.

Klasy abstrakcyjne

- ▶ Nie wszystkie metody w klasie abstrakcyjnej muszą być abstrakcyjne.
- ▶ Żaden konstruktor ani destruktor nie może być abstrakcyjny.
- ▶ Nie można utworzyć obiektu klasy abstrakcyjnej:
 - ▶ nie wolno zdefiniować funkcji, która odbierałaby argument takiej klasy przez wartość;
 - ▶ nie wolno zdefiniować funkcji, która zwracałaby wynik takiej klasy przez wartość;
 - ▶ klasa abstrakcyjna nie może być typem w jawnej konwersji.

Wirtualny destruktor

- ▶ W klasach polimorficznych (zawierających metody wirtualne) destruktor definiujemy jako wirtualny.

Konstruktor nie może być wirtualny ale...

- ▶ Czasami istnieje potrzeba wyprodukowania nowego obiektu tej samej klasy - w takiej sytuacji można zdefiniować funkcję wirtualną, która będzie przygotowywać taki obiekt (zastąpi konstruktor domyślny albo kopiujący).